VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

Araga Laxman Anirudhadithya(1BM22CS050)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
Sep-2024 to Jan-2025

B.M.S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "Machine Learning (23CS6PCMAL)" carried out by Araga Laxman Anirudhadithya(1BM22CS050)**,** who is Bonafide student of **B.M.S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum. The Lab report has been approved as it satisfies the academic requirements in respect of a Machine Learning (23CS6PCMAL) work prescribed for the said degree.

Lab Faculty Incharge Name: **Ms. Saritha A N** Assistant Professor Department of CSE, BMSCE	**Dr. Kavitha Sooda** Professor & HOD Department of CSE, BMSCE

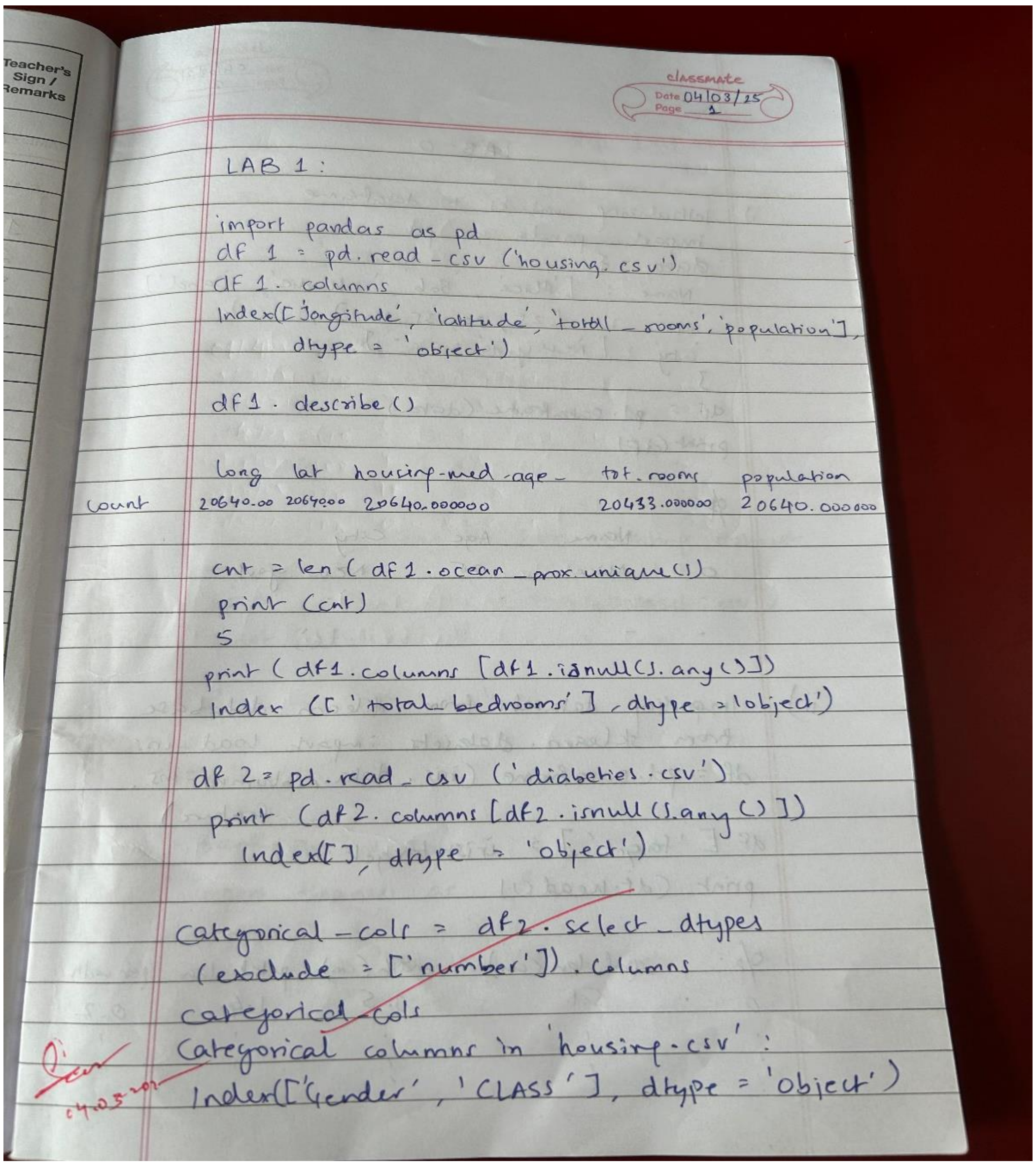
Index

Link: https://github.com/AnirudhCSE-050/ML-Lab-Sem6

Program 1

Write a python program to import and export data using Panda's library functions

Screenshot



Date 04/03/25
Page 1

LAB 1:

import pandas as pd
df 1 = pd.read_csv ('housing.csv')
df 1. columns
Index(['longitude', 'latitude', 'total_rooms', 'population'],
dtype = 'object')

df1 . describe ()

	long	lat	housing-med-age-	tot. rooms	population
count	20640.00	20640.00	20640.000000	20433.000000	20640.000000

cnt = len (df1 . ocean_prox. unique())
print (cnt)
5
print (df1.columns [df1 . isnull(). any()])
Index (['total_bedrooms'], dtype = 'object')

df 2 = pd . read_csv ('diabeties . csv')
print (df2. columns [df2 . isnull (). any ()])
Index([], dtype = 'object')

categorical_cols = df2. select_dtypes
(exclude = ['number']) . Columns
categorical_cols
Categorical columns in 'housing.csv' :
Index(['Gender', 'CLASS'], dtype = 'object')

04.03.2025

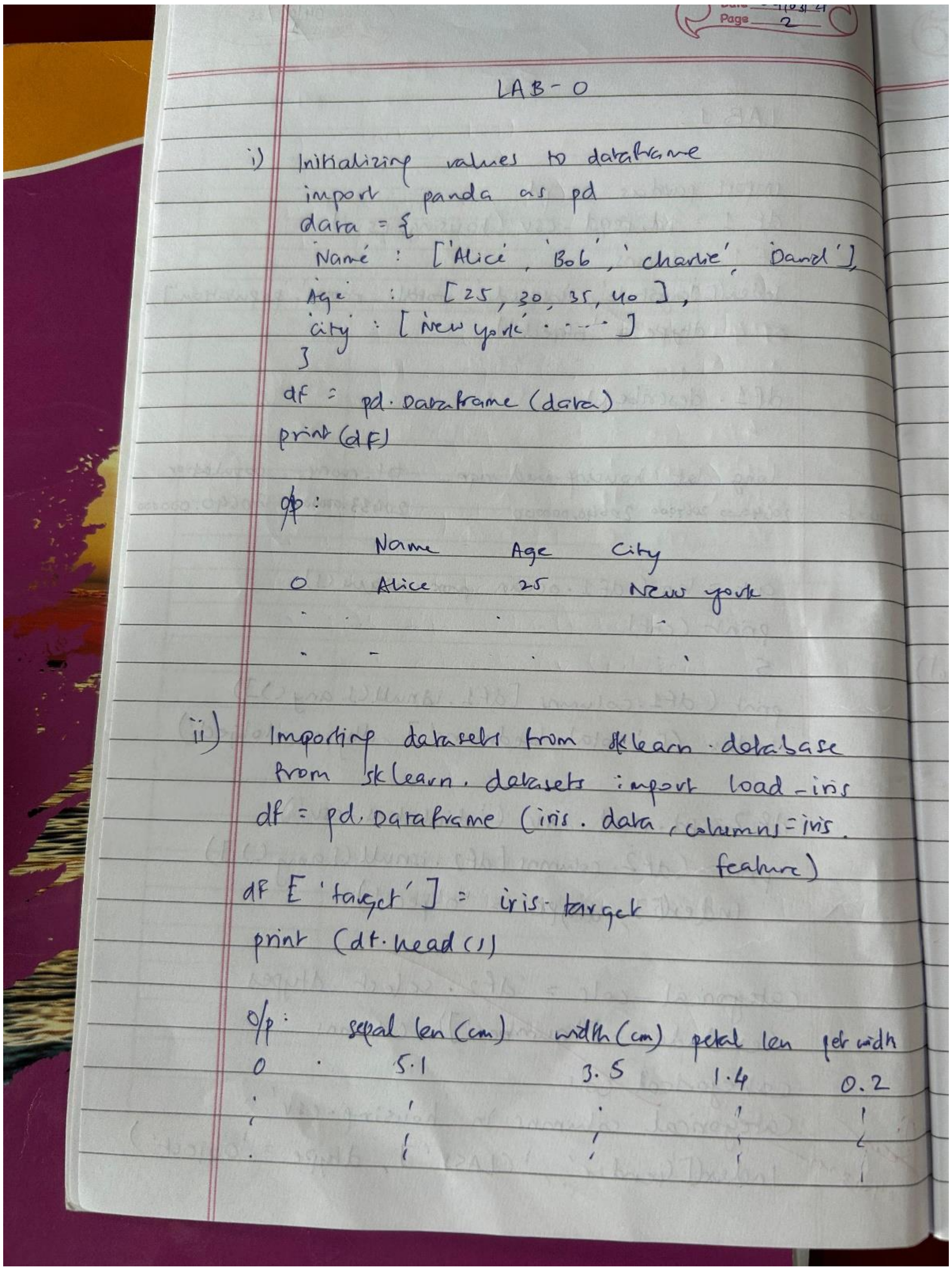
Code:

```
import pandas as pd
try:
    df = pd.read_csv('input.csv')
    print("Data imported successfully!\n")
    print(df)
except FileNotFoundError:
    print("The file 'input.csv' was not found.")
df["Processed"] = True
df.to_csv('output.csv', index=False)
print("\nData exported successfully to 'output.csv'.")
```

Program 2

Demonstrate various data pre-processing techniques for a given dataset

Screenshots:



Date 4/03/24
Page 2

LAB - 0

i) Initializing values to dataframe

```
import panda as pd
data = {
 'Name' : ['Alice', 'Bob', 'charlie', 'David'],
 'Age' : [25, 30, 35, 40],
 'city' : ['New york' ....]
}
df = pd.DataFrame (data)
print (df)
```

o/p :

	Name	Age	City
0	Alice	25	New york
.	.	.	.
.	.	.	.

ii) Importing datasets from sklearn database

```
from sklearn.datasets import load_iris
df = pd.DataFrame (iris.data, columns = iris.
                                   feature)
df ['target'] = iris.target
print (df.head())
```

o/p :

	sepal len (cm)	width (cm)	petal len	pet width
0	5.1	3.5	1.4	0.2
:	:	:	:	:
:	:	:	:	:

Code

```
import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder, StandardScaler, MinMaxScaler


data = {

    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve', None],

    'Age': [25, 30, np.nan, 35, 29, 40],
```

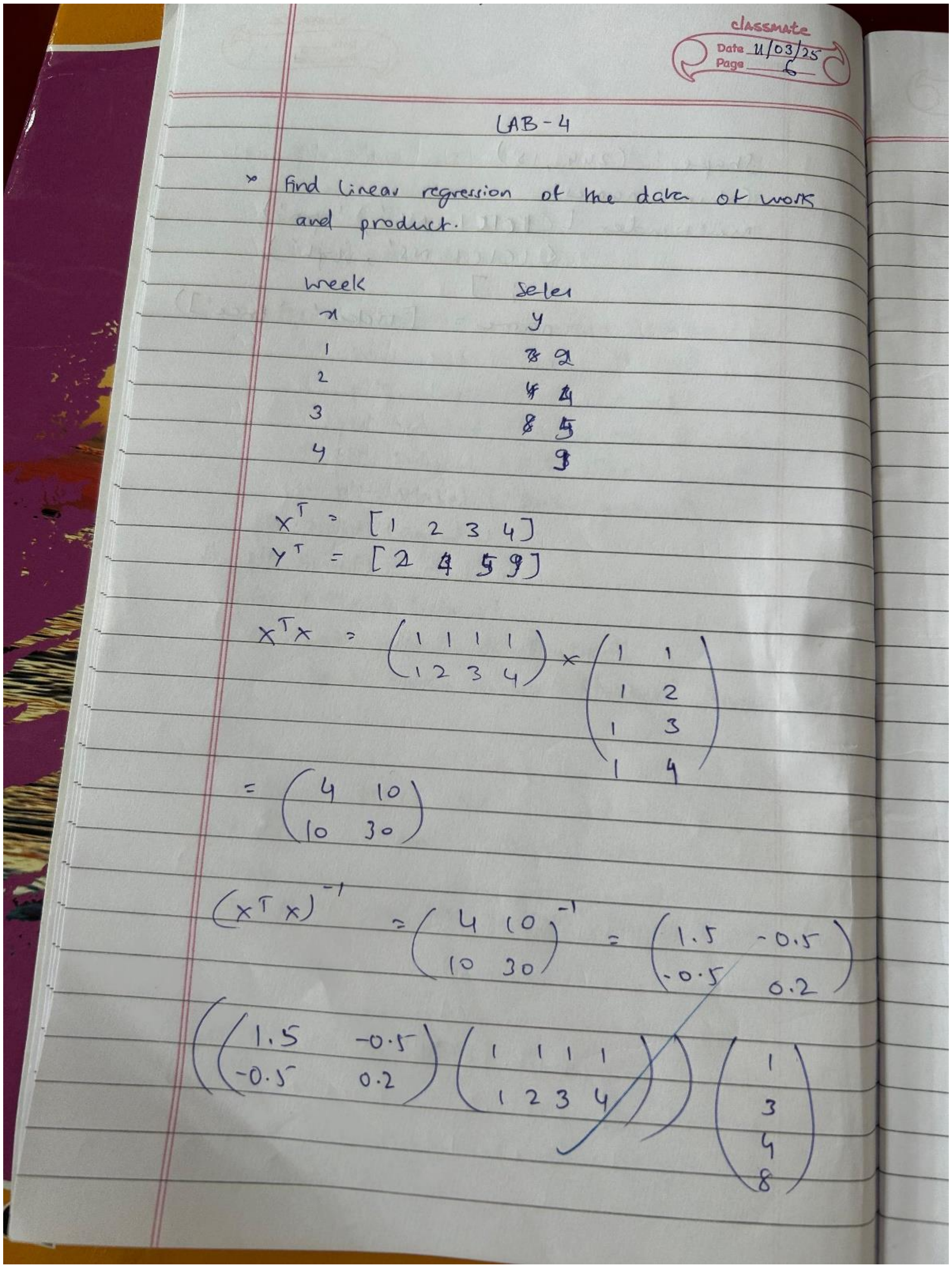
```
    'Department': ['HR', 'IT', 'Finance', 'IT', 'HR', 'Finance'],
    'Salary': [50000, 60000, 58000, 62000, np.nan, 52000]
}

df = pd.DataFrame(data)

print("Original DataFrame:\n", df)

df['Age'].fillna(df['Age'].mean(), inplace=True)
df['Salary'].fillna(df['Salary'].median(), inplace=True)
df['Name'].fillna('Unknown', inplace=True)

le = LabelEncoder()
df['Department_Encoded'] = le.fit_transform(df['Department'])

df.drop_duplicates(inplace=True)

df.rename(columns={'Salary': 'Monthly_Salary'}, inplace=True)

df['Age'] = df['Age'].astype(int)

scaler = MinMaxScaler()
df['Salary_Normalized'] = scaler.fit_transform(df[['Monthly_Salary']])

standard_scaler = StandardScaler()
```

```
df['Age_Standardized'] = standard_scaler.fit_transform(df[['Age']])

print("\nPreprocessed DataFrame:\n", df)
```

Program 3

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshots



classmate
Date 11/03/25
Page 6

LAB - 4

* Find linear regression of the data of work and product.

Week x	Seles y
1	~~3~~ 2
2	4
3	~~8~~ 5
4	9

$X^T = [1 \ 2 \ 3 \ 4]$

$Y^T = [2 \ 4 \ 5 \ 9]$

$$X^T X = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \end{pmatrix} \times \begin{pmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{pmatrix}$$

$$= \begin{pmatrix} 4 & 10 \\ 10 & 30 \end{pmatrix}$$

$$(X^T X)^{-1} = \begin{pmatrix} 4 & 10 \\ 10 & 30 \end{pmatrix}^{-1} = \begin{pmatrix} 1.5 & -0.5 \\ -0.5 & 0.2 \end{pmatrix}$$

$$\left(\begin{pmatrix} 1.5 & -0.5 \\ -0.5 & 0.2 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \end{pmatrix} \right) \begin{pmatrix} 1 \\ 3 \\ 4 \\ 8 \end{pmatrix}$$

classmate
Date
Page 14

$$\text{softmax}(z_3) = \frac{e^0}{e^2 + e^1 + e^0} = 0.091$$

$\therefore$ probabilites of 3 classes are 66.8 %, 24.4 % & 9.1 %.

18/03/2021

Code

Linear Regression

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_boston
from sklearn.linear_model import LinearRegression
```

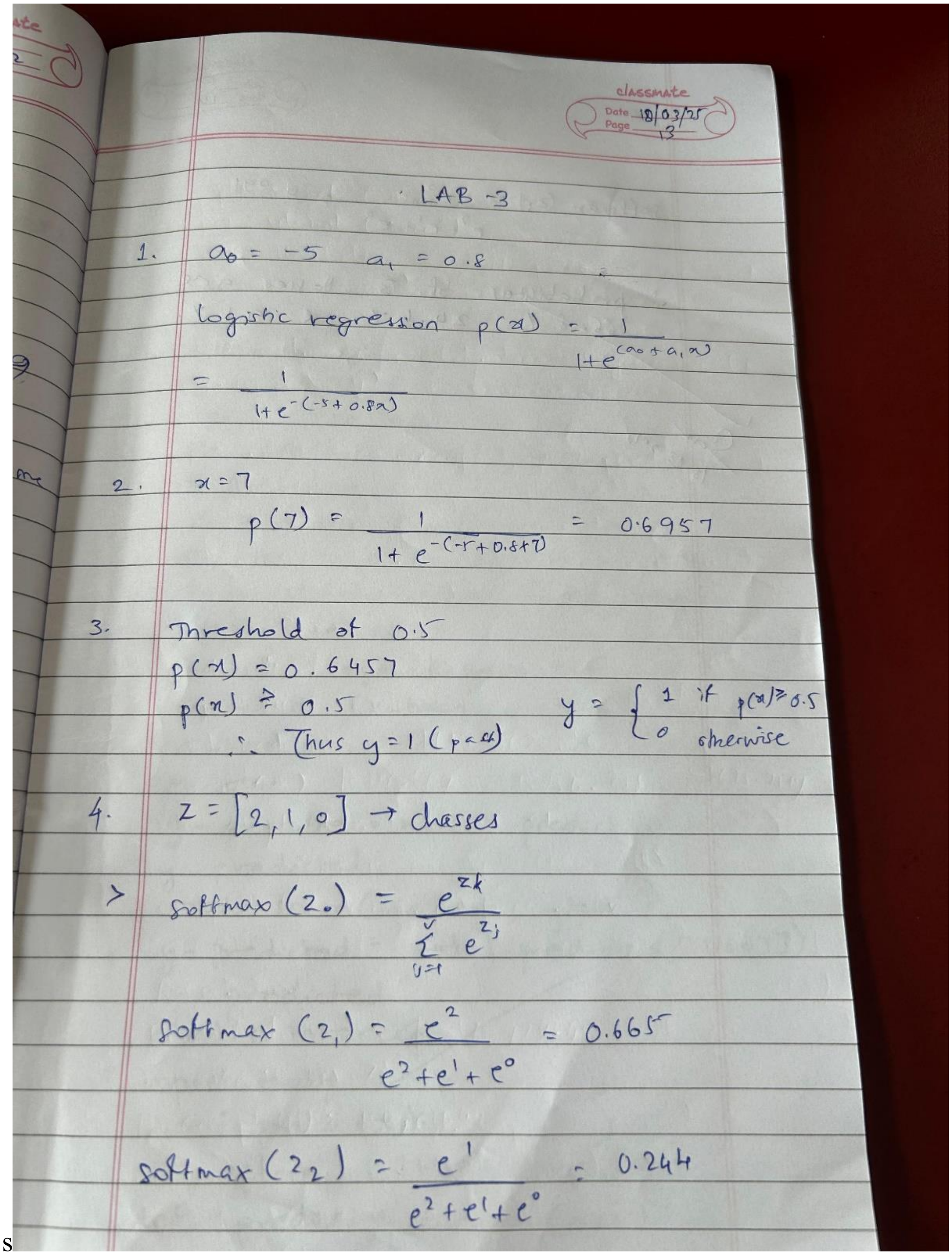
from sklearn.model_selection import train_test_split

```
from sklearn.metrics import mean_squared_error, r2_score

# Load dataset
boston = load_boston()
df = pd.DataFrame(boston.data, columns=boston.feature_names)
df['PRICE'] = boston.target

# Use only one feature for simple linear regression (e.g., RM = average number of rooms)
X = df[['RM']]
y = df['PRICE']

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)

# Train model
lr = LinearRegression()
lr.fit(X_train, y_train)
```

```
# Predict
y_pred = lr.predict(X_test)

# Output
print("Linear Regression Results")
print("Coefficients:", lr.coef_)
print("Intercept:", lr.intercept_)
print("MSE:", mean_squared_error(y_test, y_pred))
print("R² Score:", r2_score(y_test, y_pred))

# Plot
plt.scatter(X_test, y_test, color='blue')
plt.plot(X_test, y_pred, color='red')
plt.xlabel('Average Number of Rooms (RM)')
plt.ylabel('House Price')
plt.title('Simple Linear Regression')
plt.show()

# Multiple Linear Regression

# Use all features
X = df.drop('PRICE', axis=1)
y = df['PRICE']

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)

# Train model
mlr = LinearRegression()
mlr.fit(X_train, y_train)

# Predict
y_pred = mlr.predict(X_test)

# Output
print("\nMultiple Linear Regression Results")
print("Coefficients:", mlr.coef_)
print("Intercept:", mlr.intercept_)
print("MSE:", mean_squared_error(y_test, y_pred))
print("R² Score:", r2_score(y_test, y_pred))
```

Program 4

Build Logistic Regression Model for a given dataset



LAB -3

1. $a_0 = -5 \quad a_1 = 0.8$

logistic regression $p(x) = \frac{1}{1+e^{-(a_0 + a_1 x)}}$

$= \frac{1}{1+e^{-(-5+0.8x)}}$

2. $x = 7$

$p(7) = \frac{1}{1+e^{-(-5+0.8*7)}} = 0.6957$

3. Threshold of 0.5

$p(x) = 0.6457$

$p(x) \geq 0.5$

$\therefore$ Thus $y = 1$ ($p \geq 0.5$)

$y = \begin{cases} 1 & \text{if } p(x) \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$

4. $Z = [2, 1, 0] \rightarrow$ classes

> $\text{softmax}(z_0) = \frac{e^{z_k}}{\sum_{j=1}^{V} e^{z_j}}$

$\text{softmax}(z_1) = \frac{e^2}{e^2+e^1+e^0} = 0.665$

$\text{softmax}(z_2) = \frac{e^1}{e^2+e^1+e^0} = 0.244$

Screenshot's

Code

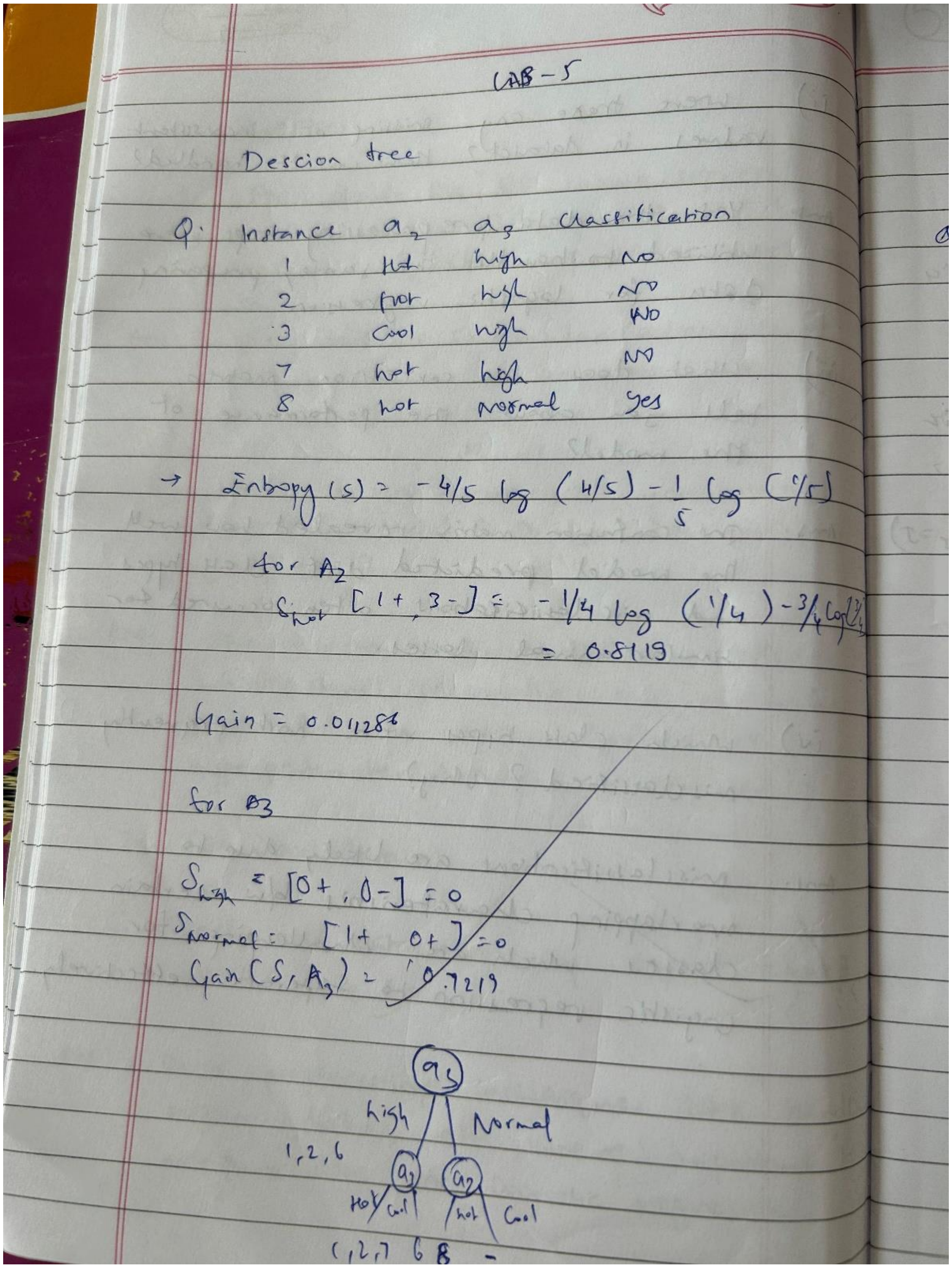
```
import pandas as pd

from sklearn.datasets import load_iris

from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import train_test_split

from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['species'] = iris.target
```

```
df_binary = df[df['species'] != 2]  # Remove class 2 (Virginica)

X = df_binary.iloc[:, :-1]   # Features
y = df_binary['species']      # Target (0 or 1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = LogisticRegression()
model.fit(X_train, y_train)

# Step 5: Predict and evaluate
y_pred = model.predict(X_test)

print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nAccuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

Program 5

Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample

Screenshots



LAB – 5

Descion tree

Q.

Instance	a_2	a_3	Classification
1	Hot	high	No
2	Hot	high	No
3	Cool	high	No
7	hot	high	No
8	hot	Normal	yes

→ $Entropy(S) = -4/5 \log(4/5) - \frac{1}{5} \log(1/5)$

for A_2

$S_{hot} [1+, 3-] = -1/4 \log(1/4) - 3/4 \log(3/4)$

$= 0.8119$

Gain = 0.011286

for A_3

$S_{high} = [0+, 0-] = 0$

$S_{Normal}: [1+ \quad 0+] = 0$

$Gain(S, A_3) = 0.7219$

Page 19

$\alpha_1 \hat{s}_1 \hat{s}_1 + \alpha_2 \hat{s}_2 \hat{s}_1 + \alpha_3 \hat{s}_3 \hat{s}_1 = +1$
$\alpha_1 \hat{s}_1 \hat{s}_2 + \alpha_2 s_2 s_2 + \alpha_3 s_3 s_2 = +1$
$\alpha_1 s_1 \hat{s}_3 + \alpha_2 s_2 s_3 + \alpha_3 s_3 s_3 = -1$

after sub.

$\alpha_1 = 13/4$

$\alpha_2 = 13/4$

$\alpha_3 = -7/2$

$w = \alpha_1 \hat{s}_1 + \alpha_2 \hat{s}_2 + \alpha_3 \hat{s}_3$

$= -\begin{bmatrix} +1 \\ 0 \end{bmatrix}$ $\quad b = -3$

$b + 3 = 0$

intercept on x axis $= 3$

$= \begin{bmatrix} 1 \\ 0 \end{bmatrix} \rightarrow$ line parallel to y axis.

Code

```
import pandas as pd
```

```
fromsklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
data= {
  'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rain', 'Rain', 'Rain', 'Overcast',
          'Sunny', 'Sunny', 'Rain', 'Sunny', 'Overcast', 'Overcast', 'Rain'],
  'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool',
            'Mild', 'Cool', 'Mild', 'Mild', 'Mild', 'Hot', 'Mild'],
  'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal', 'Normal',
          'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal', 'High'],
  'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong',
        'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak', 'Strong'],
  'PlayTennis': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes',
           'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']
}

df = pd.DataFrame(data)
le = LabelEncoder()
for column in df.columns:
  df[column] = le.fit_transform(df[column])

# Step 3: Separate features and label
X = df.drop('PlayTennis', axis=1)
y = df['PlayTennis']
clf = DecisionTreeClassifier(criterion='entropy')  # ID3 uses 'entropy'
```

```
clf = clf.fit(X, y)

print("\nDecision Tree Rules:")

tree_text =tree.export_text(clf, feature_names=X.columns.tolist())

print(tree_text)

# Example: Outlook=Rain, Temperature=Mild, Humidity=High, Wind=Weak

# Encode input sample with same label encoding order used earlier

sample = pd.DataFrame({

    'Outlook': [le.transform(['Rain'])[0]],

    'Temperature': [le.transform(['Mild'])[0]],

    'Humidity': [le.transform(['High'])[0]],

    'Wind': [le.transform(['Weak'])[0]]

})


# Predict

prediction = clf.predict(sample)

result = 'Yes' if prediction[0] == 1 else 'No'

print(f"\nPrediction for new sample (Rain, Mild, High, Weak): {result}")
```

<u>Program 6</u> Build KNN Classification model for a given dataset

Page 24

LAB - 6

KNN :

Q.

Person	age	sal.	target	distance
A	18	50	N	52.8
B	23	55	N	46.57
C	24	70	N	31.95
D	41	60	Y	40.44
E	43	70	Y	31.04
F	38	60	Y	60.07
X	35	100	Y	?

X = (35, 100)

k = 3

Dist	Rank	Target
31.04	1	Y
31.95	2	N
40.44	3	Y

x (35, 100) target will be Y.

Code

```
import pandas as pd
fromsklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
```

```
fromsklearn.neighbors import KNeighborsClassifier

fromsklearn.metrics import accuracy_score, classification_report, confusion_matrix


# Step 1: Load the Iris dataset

iris = load_iris()

X = pd.DataFrame(iris.data, columns=iris.feature_names)
```

```
y = pd.Series(iris.target)


# Step 2: Split the data (80% training, 20% testing)
X_train, X_test, y_train, y_test =train_test_split(X, y, test_size=0.2, random_state=42)


# Step 3: Build KNN model (k = 3)
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)


# Step 4: Predict and evaluate
y_pred = knn.predict(X_test)


# Results
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("Accuracy Score:", accuracy_score(y_test, y_pred))
```

Program 7

Build Support vector machine model for a given dataset

Screenshots

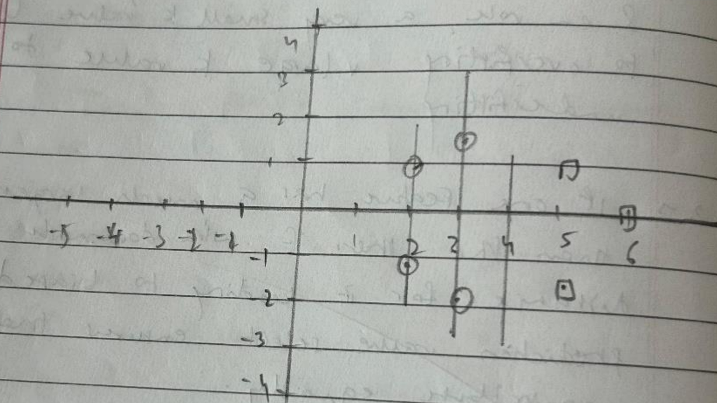
Date 08/04/25
Page 28

LAB 7:

Draw an optimal hyperplane using linear svm to classify points.

$\{(1,1), (2,1), (1,-1), (2,-1)\} \rightarrow$ +ve labels

$\{(4,0), (5,1), (5,-1), (6,0)\} \rightarrow$ -ve labels



$s_1 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$ $s_2 = \begin{bmatrix} 2 \\ -1 \end{bmatrix}$ $s_3 = \begin{bmatrix} 4 \\ 0 \end{bmatrix}$

$\bar{s}_1 = \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix}$ $\bar{s}_2 = \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}$ $\bar{s}_3 = \begin{bmatrix} 4 \\ 0 \\ 1 \end{bmatrix}$

Code

```
import pandas as pd
fromsklearn.datasets import load_iris
fromsklearn.model_selection import train_test_split
from sklearn.svm import SVC
fromsklearn.metrics import confusion_matrix, classification_report, accuracy_score

# Step 1: Load the Iris dataset
iris = load_iris()
X = pd.DataFrame(iris.data, columns=iris.feature_names)
y = pd.Series(iris.target)

# Step 2: Split the data into train and test sets (80% train, 20% test)
```

```
X_train, X_test, y_train, y_test =train_test_split(X, y, test_size=0.2, random_state=42)


# Step 3: Build and train the SVM model(linear kernel)
svm_model = SVC(kernel='linear')
svm_model.fit(X_train, y_train)


# Step 4: Predict and evaluate
y_pred = svm_model.predict(X_test)


# Output results
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("Accuracy Score:", accuracy_score(y_test, y_pred))
```

Program 8

Implement Random Forest ensemble method on a given dataset

Screenshots

Date 25/04/24
Page 30

LAB 8:

* Descision tree	Random forest
1. Simple tree	1. Multiple
2. Risk accurate	2. More accurate
3. Fast to train	3. Slow to train
4. Single prediction	4. Majority vote

2. Parameters of Random forest clss.

→ n. estimate no. of trees in forest
→ max depth & min depth of tree.

Algo:

1. Training dataset:

2. for n times:
   - Randomly select samples w replacement
   - grow a decision tree
   - split nodes using best features.

3. Aggregate Pred:
   - Class^n - Maj. vote
   - Regression - Avg.

4. O/p prediction

Page 31

Code:

```
import pandas as pd
from sklearn _ selection import train_test_split
from sklearn. ensemble import Randomforest classifier
from sklearn. matrix import accuracy. score.

x_train , y_train , x_test , y_test = train.
test. split (x, y , test. size = 0.2 random_state = 42)

rf _default = Random forest classification (n_estimators =
  10 , random_state = 42)
rf_ default - fix (x_train, y_train)
y_pred _default = rf . default predict (x_test)
accuracy _default = accuracy _score (y_test , y_pred)
print (accuracy_default)

estimator = [10 , 50, 100, 200, 500]
scores = [ ]

f n estimator
  rf = RandomForestClassification (n_estimators = n ,
  random_state = 42 )
  rf _ fit (x_train, y_train)
   y_pred = rf . predict (x_test)
    score = accuracy. score (y_test , y_pred)
    scores . append(score)
     print(score)
```

Code

```
import pandas as pd

fromsklearn.datasets import load_iris

fromsklearn.model_selection import train_test_split

fromsklearn.ensemble import RandomForestClassifier

fromsklearn.metrics import classification_report, confusion_matrix, accuracy_score


# Step 1: Load Iris dataset

iris = load_iris()

X = pd.DataFrame(iris.data, columns=iris.feature_names)

y = pd.Series(iris.target)
```

```
# Step 2: Split into train and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test =train_test_split(X, y, test_size=0.2, random_state=42)

# Step 3: Train Random Forest model
rf = RandomForestClassifier(n_estimators=100, random_state=42)  # 100 trees
rf.fit(X_train, y_train)

# Step 4: Predict and evaluate
y_pred = rf.predict(X_test)

print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("Accuracy Score:", accuracy_score(y_test, y_pred))
```

Program 9

Implement Boosting ensemble method on a given dataset

Screenshots

LAB-9 :

Boosting:
Combines multiple weak learners to create a ~~strr~~ strong learner. It works by training modules sequential where each match one ~~sum~~ fun.

Parameters:

estimator The base model
n-estimator no. of weak learners.
learning-rate shrinks contribution of each learner
algo 'SAMME' or 'SAMME.R'
random-state for prediction

Algorithm.

1. Start w eq. wts for all training for each sample
2. train a weak model
3. calc. error & update sample wt.
4. Add weak model to ensemble w c wt based on its accuracy.
5. Repeat for n-estimators
6. Final prediction.

Code

```
import pandas as pd
fromsklearn.datasets import load_iris
fromsklearn.model_selection import train_test_split
fromsklearn.ensemble import AdaBoostClassifier
fromsklearn.metrics import classification_report, confusion_matrix, accuracy_score
fromsklearn.tree import DecisionTreeClassifier

# Step 1: Load the Iris dataset
iris = load_iris()
```

```
X = pd.DataFrame(iris.data, columns=iris.feature_names)

y = pd.Series(iris.target)


# Step 2: Split into train and test sets (80% train, 20% test)

X_train, X_test, y_train, y_test =train_test_split(X, y, test_size=0.2, random_state=42)


# Step 3: Build AdaBoost modelwith DecisionTreeClassifier as base estimator

base_estimator = DecisionTreeClassifier(max_depth=1)

model= AdaBoostClassifier(base_estimator=base_estimator, n_estimators=50, learning_rate=1.0,
random_state=42)

model.fit(X_train, y_train)


# Step 5: Predict and evaluate

y_pred = model.predict(X_test)

print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

print("\nClassification Report:\n", classification_report(y_test, y_pred))

print("Accuracy Score:", accuracy_score(y_test, y_pred))
```

Program 10

Build k-Means algorithm to cluster a set of data stored in a .CSV file

Screenshots

Code

```
import pandas as pd
fromsklearn.cluster import KMeans
import matplotlib.pyplot as plt
fromsklearn.preprocessing import StandardScaler

# Step 1: Load dataset from CSV
df = pd.read_csv('your_dataset.csv')  # Replace with your file path

# Optional: View first few rows
print("Data Preview:\n", df.head())
```

```
# Step 2: Select relevant numeric columns for clustering

# You can specify specific columns like: df[['column1', 'column2']]

X = df.select_dtypes(include='number')


#Step 3: Scale thedata(important for K-Means)

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)


# Step 4: Apply K-Means clustering (e.g., 3 clusters)

kmeans = KMeans(n_clusters=3, random_state=42)

df['Cluster'] = kmeans.fit_predict(X_scaled)


# Step 5: Print cluster centers

print("Cluster Centers:\n", kmeans.cluster_centers_)


#Optional Step 6: Visualize(works well for 2D or PCA-reduced data)

if X.shape[1] >= 2:

    plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=df['Cluster'], cmap='viridis')

    plt.title("K-Means Clustering")

    plt.xlabel("Feature 1")

    plt.ylabel("Feature 2")

    plt.show(
```

Date
Page 31

Code:

```
import pandas as pd
from sklearn _ selection import train test split
from sklearn. ensemble import Randomforest classifica
from sklearn. matrix import accuracy. score.

x_train , y-train , x_test , y_test = train.
test. split (x, y , test . size = 0.2 random-state = 42)

rf _ default = Random forest classification (n-estimators =
  10 , random - state = 42)
rf- default . fix (x_train, y- train)
y-pred_ default = rf . default predict (x-test)
accuracy _default = accuracy - score (y-test , y-pred)
print (accuracy - default)

estimator = [10 , 50, 100, 200, 500]
scores = []

f n estimator:
  rf = RandomForestClassification (n-estimators =n,
  random-state = 42)
  rf - fit (x_train, y-train)
  y- pred = rf . predict (x-test)
  score = accuracy .score (y-test, y-pred)
  scores . append (score)
  print (score)
```

Program 11

Implement Dimensionality reduction using Principal Component Analysis (PCA) method

Screenshots

Date __/__/__
Page ____

Lab 10 : PCA

1. Calculate Mean
2. Calculate covariance matrix
3. Eigen val of covariance matrix
4. Computation of eigen vectors unit eigen vectors
5. Computation of first principle component
6. Geometric meaning of 1st principle component

Ex 1	Ex 2	Ex 3	Ex 4	Ex 5
4	8	13	13	7
11	4	5	5	14

calculate mean

$$\bar{x_1} = 1/4\,(4 + 7 + 13 + \ldots)$$

$$\bar{x_2} = 1/4\,(11 + 4 + 5 + 14) = 8.5$$

$$\text{cov}(x_1, x_2) = 1/N \sum_{i=1}^{N} (x_{2k} - x_2)^2$$

$$= 1/3\,[(14-8) + (11-8)^2 + (8-8)^2$$
$$(4-8.5)^2\ (5-8.5) + (3-8)(4.4)$$
$$= -11$$

$$S = \begin{bmatrix} \text{cov}(x_1\ x_2) & \text{cov}(x_1\ x_2) \\ \text{cov}(x\ x_1) & \text{cov}(x\ x_1) \end{bmatrix}$$

$$S = \begin{bmatrix} 14 & -4 \\ 11 & 23 \end{bmatrix}$$

iii)

Eigen value = $\begin{bmatrix} 14-\lambda & -1 \\ -11 & 23-\lambda \end{bmatrix}$

Code

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.decomposition import PCA

import matplotlib.pyplot as plt


# Step 1: Load the Iris dataset

iris = load_iris()

X = iris.data
```

```
y = iris.target
feature_names = iris.feature_names

# Step 2: Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 3: Apply PCA (reduce to 2 components for visualization)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Step 4: Plot the 2D PCA result
plt.figure(figsize=(8, 6))
scatter = plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='viridis', edgecolor='k', s=60)
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.title("PCA - Iris Dataset")
plt.legend(handles=scatter.legend_elements()[0], labels=iris.target_names)
plt.grid(True)
plt.show()

# Explained variance
print("Explained variance ratio:", pca.explained_variance_ratio_)
```